On the Performance of k-ary n-cube Interconnection Networks

William J Dally

Computer Science Department
California Institute of Technology

5228:TR:86

On The Performance of k-ary n-cube Interconnection Networks[1]

5228:TR:86

William J. Dally[2]

Department of Computer Science
California Institute of Technology
Pasadena, California 91125

Abstract

The performance of k-ary n-cube interconnection networks is analyzed under the assumption of constant wire bisection. It is shown that low-dimensional k-ary n-cube networks (e.g., tori) have lower latency and higher hot-spot throughput than high-dimensional networks (e.g., binary n-cubes) with the same bisection width.

1 Introduction

The critical component of a concurrent computer is its communication network. Many algorithms are communication rather than processing limited. The performance of these algorithms depends on high-throughput low-latency communication. In this paper we investigate the class of k-ary n-cube interconnection networks and show that under an assumption of constant wire density, low-dimensional networks out perform high-dimensional networks.

As the grain size of concurrent computers continues to decrease, communication latency becomes a more important factor. As grain size shrinks, the diameter of the machine grows, messages occur more frequently, and fewer instructions are executed in response to each message. To execute fine grain concurrent programs efficiently, communication networks must be designed to minimize latency.

Network topology is discussed in Section 2. The family of k-ary n-cube networks is described. Section 3 introduces *wormhole routing* [16], a low-latency routing technique. Network cost is determined primarily by wire density. Section 4 introduces the idea of *bisection*

[1]The research described in this paper was sponsored in part by the Defense Advanced Research Projects Agency, ARPA Order number 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597, in part by Intel Corporation, and in part by an AT&T Ph.D. fellowship.

[2]current address: Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts 02139

width, and discusses delay models for network channels. Finally in Section 5 the performance of these networks is calculated under the assumption of constant wire density. It is shown that low-dimensional networks achieve lower latency and better hot-spot throughput than do high-dimensional networks and that total network throughput is independent of dimension.

2 k-ary n-cubes

Many different network topologies have been proposed for use in concurrent computers: trees [4] [10] [17], Benes networks[3], Batcher sorting networks [1], shuffle exchange networks [18], *Omega* networks [9], *indirect* binary n-cube or *flip* networks [2] [14], and direct binary n-cubes [15], [12], [19]. The binary n-cube is a special case of the family of k-ary n-cubes, cubes with n dimensions and k nodes in each dimension.

Most concurrent computers have been built using networks that are either k ary n-cubes or are isomorphic to k-ary n-cubes: meshes, tori, direct and indirect binary n-cubes, and Omega networks. Thus, in this paper we restrict our attention to k-ary n-cube networks. It is the dimension of the network that is important, not the details of its topology. We refer to n as the *dimension* of the cube and k as the *radix*. Dimension, radix, and number of nodes are related by the equation

$$N = k^n, \quad \left(k = \sqrt[n]{N}, \quad n = \log_k N\right). \tag{1}$$

A node in a k-ary n-cube can be identified by an n-digit radix k address, $a_0, \ldots, a_{n-1}$. The i^{th} digit of the address, a_i, represents the nodes position in the i^{th} dimension. Each node can send messages to its upper neighbor in each dimension, i, with address, $a_0, \ldots, a_i + 1 (\mathrm{mod} k), \ldots, a_{n-1}$.

In this paper we assume that our k-ary n-cube are unidirectional for simplicity. We will see that our results do not change appreciably for bidirectional networks. For an actual machine, however, there are many compelling reasons to make our networks bidirectional. Most importantly, bidirectional networks allow us to exploit locality of communication. If an object, A, sends a message to an object, B, there is a high probability of B sending a message back to A. In a bidirectional network, a round trip from A to B can be made short by placing A and B close together. In a unidirectional network, a round trip will always involve completely circling the machine in at least one dimension.

Figures 1-3 show three k-ary n-cube networks in order of decreasing dimension. Figure 1 shows a binary 6-cube (64 nodes). A 3-ary 4-cube (81 nodes) is shown in Figure 2. An 8-ary 2-cube (64 nodes), or torus, is shown in Figure 3. Each line in Figure 1 represents two communication channels, one in each direction, while each line in Figures 2 and 3 represents a single communication channel.

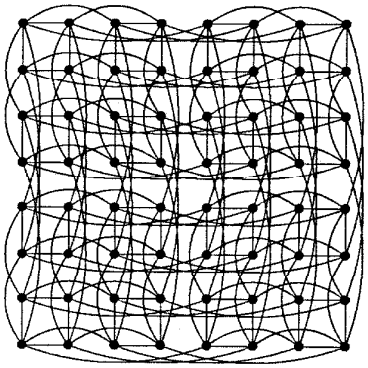

Figure 1: A Binary 6-Cube Embedded in the Plane

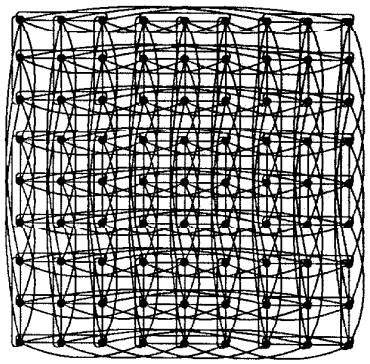

Figure 2: A Ternary 4-Cube Embedded in the Plane

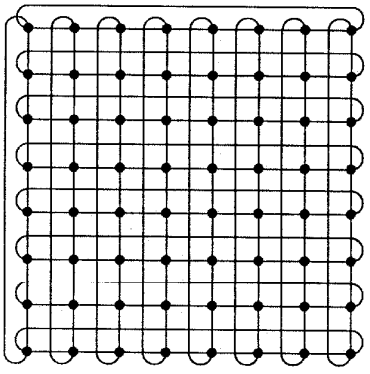

Figure 3: An 8-ary 2-Cube (Torus)

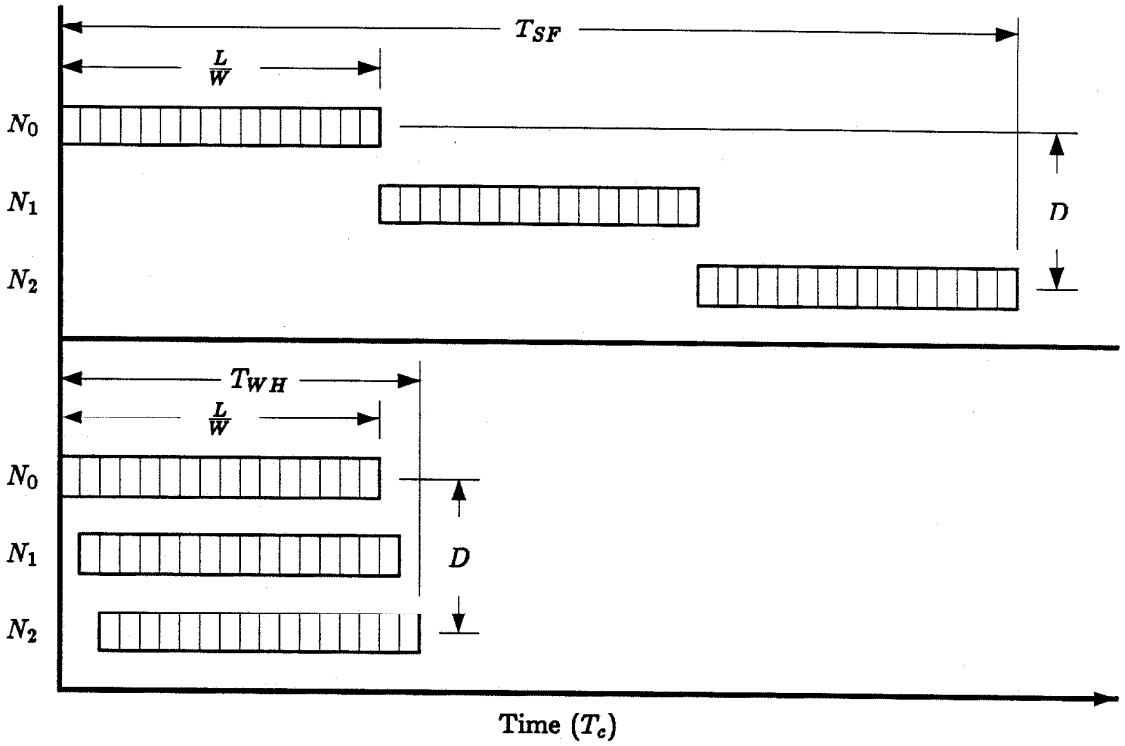

Figure 4: Latency of store-and-forward routing (top) vs. wormhole routing (bottom).

3 Wormhole Routing

In this paper we consider networks that use *wormhole*[16] rather than *store-and-forward* [20] routing. Instead of storing a packet completely in a node and then transmitting it to the next node, wormhole routing operates by advancing the head of a packet directly from incoming to outgoing channels. Only a few flow control digits (flits) are buffered at each node. A *flit* is the smallest unit of information that a queue or channel can accept or refuse.

As soon as a node examines the header flit(s) of a message, it selects the next channel on the route and begins forwarding flits down that channel. As flits are forwarded, the message becomes spread out across the channels between the source and destination. It is possible for the first flit of a message to arrive at the destination node before the last flit of the message has left the source. Because most flits contain no routing information, the flits in a message must remain in contiguous channels of the network and cannot be interleaved with the flits of other messages. When the header flit of a message is blocked, all of the flits of a message stop advancing and block the progress of any other message requiring the channels they occupy.

A method similar to wormhole routing, called *virtual cut-through*, is described in [8]. Virtual cut-through differs from wormhole routing in that it buffers messages when they block, removing them from the network. With wormhole routing, blocked messages remain in the network.

Figure 4 illustrates the advantage of wormhole routing. There are two components of latency, distance and message aspect ratio. The distance, D, is the number of *hops* required to get from the source to the destination. The message aspect ratio (message length, L, normalized to the channel width, W) is the number of channel cycles required to transmit the message across one channel. The top half of the figure shows store-and-forward routing. The message is is entirely transmitted from node N_0 to node N_1, then from N_1 to N_2 and so on. With store-and-forward routing, latency is the product of D, and $\frac{L}{W}$.

$$T_{SF} = T_c \left(D \times \frac{L}{W} \right). \tag{2}$$

The bottom half of Figure 4 shows wormhole routing. As soon as a flit arrives at a node, it is forwarded to the next node. With wormhole routing latency is reduced to the sum of D and $\frac{L}{W}$.

$$T_{WH} = T_c \left(D + \frac{L}{W} \right). \tag{3}$$

In both of these equations, T_c is the channel cycle time, the amount of time required to perform a transaction on a channel.

4 VLSI Complexity

VLSI computing systems [11] are wire-limited; the complexity of what can be constructed is limited by wire density, the speed at which a machine can run is limited by wire delay, and the majority of power consumed by a machine is used to drive wires. Thus, machines must be organized both logically and physically to keep wires short by exploiting locality wherever possible. The VLSI architect must organize a computing system so that its form (physical organization) fits its function (logical organization).

Networks have traditionally been analyzed under the assumption of constant channel bandwidth. Under this assumption each channel is one bit wide ($W = 1$) and has unit delay ($T_c = 1$). The constant bandwidth assumption favors networks with high dimensionality (e.g., binary n-cubes) over low-dimensional networks (e.g., tori). This assumption, however, is not consistent with the properties of VLSI technology. Networks with many dimensions require more and longer wires than do low-dimensional networks. Thus, high-dimensional networks cost more and run more slowly than low-dimensional networks. A realistic comparison of network topology must take both wire density and wire length into account.

To account for wire density, we will use bisection width [21] as a measure of network cost. The bisection width of a network is the minimum number of wires cut when the network is divided into two equal halves. Rather than comparing networks with constant channel width, W, we will compare networks with constant bisection width. Thus, we will compare low-dimensional networks with large W with high-dimensional networks with small W.

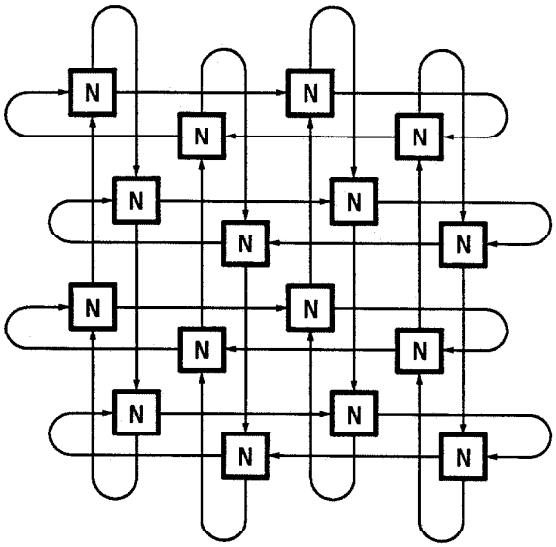

Figure 5: A Folded Torus System

The delay of a wire depends on its length, l. For short wires, the delay, t_s, is limited by charging the capacitance of the wire and varies logarithmically with wire length.

$$t_s = \tau_{\text{inv}} e \log_e Kl, \tag{4}$$

where τ_{inv} is the inverter delay, and K is a constant depending on capacitance ratios.

For long wires, delay, t_l, is limited by the speed of light.

$$t_l = \frac{l\sqrt{\epsilon_r}}{c} \tag{5}$$

In this paper we will consider three delay models: constant delay, T_c independent of length, logarithmic delay, $T_c \propto \log l$, and linear delay, $T_c \propto l$. Our main result, that latency is minimized by low-dimensional networks, is supported by all three models.

5 Performance

In this section we compare the performance of unidirectional k-ary n-cube interconnection networks using the following assumptions:

- Networks must be embedded into the plane[3].
- Nodes are placed systematically by embedding $\frac{n}{2}$ logical dimensions in each of the two physical dimensions. We assume that both n and k are even integers. The long

[3]If a three-dimensional packaging technology becomes available, the comparison changes only slightly.

end-around connections shown in Figure 3 can be avoided by folding the network as shown in Figure 5.

- For networks with the same number of nodes, *wire density is held constant.* Each network is constructed with the same bisection width, B, the total number of wires crossing the midpoint of the network. To keep the bisection width constant, we vary the width, W, of the communication channels. We normalize to the bisection width of a bit-serial ($W = 1$) binary n-cube.
- The networks use *wormhole* routing.
- No more than a single bit is in transit on any wire at a given time.
- Channel delay, T_c, is a function of wire length, l. We begin by considering channel delay to be constant. Later, the comparison is performed for both logarithmic and linear wire delays; $T_c \propto \log l$ and $T_c \propto l$.

When k is even, the channels crossing the midpoint of the network are all in the highest dimension. For each of the $\sqrt{N}$ rows of the network, there are $k^{(\frac{n}{2}-1)}$ of these channels in each direction for a total of $2\sqrt{N}k^{(\frac{n}{2}-1)}$ channels. Thus, the bisection width, B, of a k-ary n-cube with W-bit wide communication channels is

$$B(k,n) = 2W\sqrt{N}k^{(\frac{n}{2}-1)} = \frac{2WN}{k}. \tag{6}$$

For a binary n-cube, $k = 2$, the bisection width is $B(2,n) = WN$. We set B equal to N to normalize to a binary n-cube with unit width channels, $W = 1$. The channel width, $W(k,n)$, of a k-ary n-cube with the same bisection width, B, follows from (6):

$$\begin{aligned} \tfrac{2W(k,n)N}{k} &= N, \\ W(k,n) &= \tfrac{k}{2}. \end{aligned} \tag{7}$$

The peak wire density is greater than the bisection width in networks with $n > 2$ because the lower dimensions contribute to wire density. The maximum density, however, is bounded by

$$\begin{aligned} D_{\max} &= 2W\sqrt{N}\sum_{i=0}^{\frac{n}{2}-1} k^i = k\sqrt{N}\sum_{i=0}^{\frac{n}{2}-1} k^i = k\sqrt{N}\left(\frac{k^{\frac{n}{2}}-1}{k-1}\right) \\ &= k\sqrt{N}\left(\frac{\sqrt{N}-1}{k-1}\right) < \left(\frac{k}{k-1}\right)B. \end{aligned} \tag{8}$$

A plot of wire density as a function of position for one row of a binary 20-cube is shown in Figure 6. The density is very low at the edges of the cube and quite dense near the center. The peak density for the row is 1364 at position 341. Compare this density with

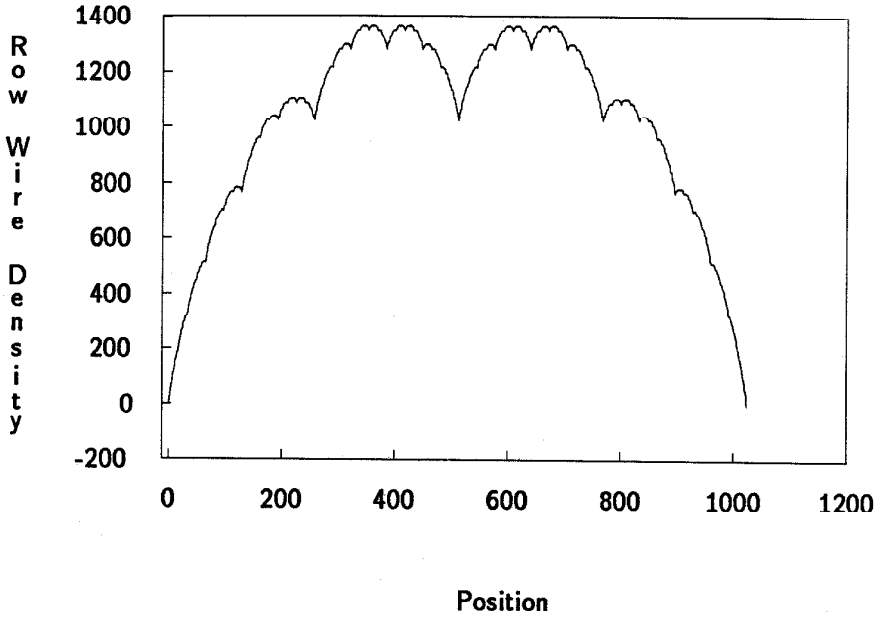

Figure 6: Wire Density vs. Position for One Row of a Binary 20-Cube

the bisection width of the row, which is 1024. In contrast, a two-dimensional torus has a wire density of 1024 independent of position. One advantage of high-radix networks is that they have a very uniform wire density. They make full use of available area.

Each processing node connects to $2n$ channels (n input and n output) each of which is $\frac{k}{2}$ bits wide. Thus, the number of pins per processing node is

$$N_p = nk. \tag{9}$$

A plot of pin density as a function of dimension for $N = 256$, 16K and 1M nodes[4] is shown in Figure 7. Low dimensional networks have the disadvantage of requiring many pins per processing node. A two-dimensional network with 1M nodes (not shown) requires 2048 pins and is clearly unrealizable. However, the number of pins decreases very rapidly as the dimension, n, increases. Even for 1M nodes, a dimension 4 node has only 128 pins. All of the configurations that give low latency also give a reasonable pin count.

5.1 Latency

Latency, T_l, is the sum of the latency due to the network and the latency due to the

[4] $1K = 1024$ and, $1M = 1K \times 1K = 1048576$.

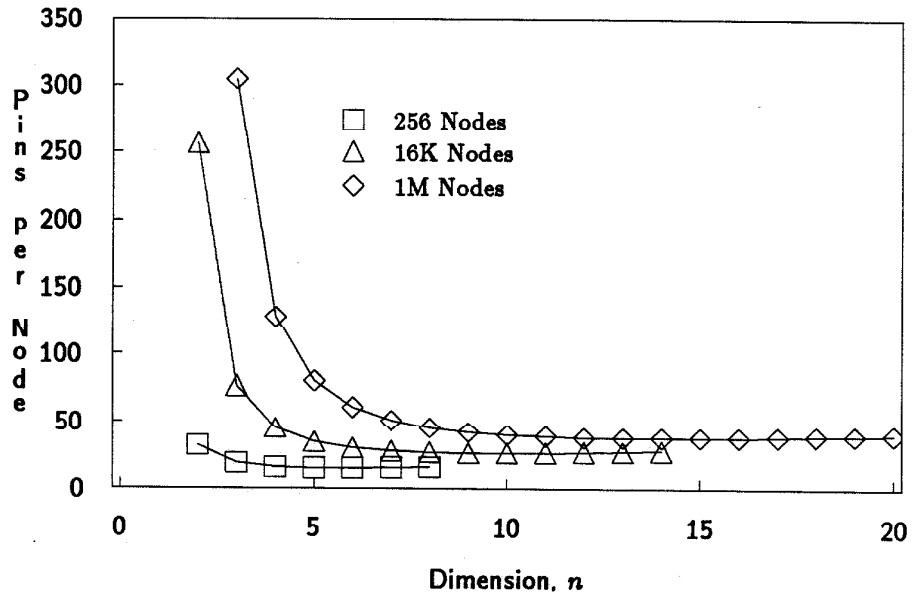

Figure 7: Pin Density vs. Dimension for 256, 16K, and 1M Nodes

processing node,

$$T_l = T_{\text{net}} + T_{\text{node}}. \tag{10}$$

In this paper we are concerned only with T_{net}. Techniques to reduce T_{node} are described in [5].

If we select two processing nodes, P_i, P_j, at random, the average number of channels that must be traversed to send a message from P_i to P_j is given by

$$D(k,n) = \left(\frac{k-1}{2}\right) n. \tag{11}$$

The average latency of a k-ary n-cube is calculated by substituting (7) and (11), into (3)

$$T_{\text{net}} = T_c \left(\left(\frac{k-1}{2}\right) n + \frac{2L}{k} \right). \tag{12}$$

Figure 8 shows the average network latency, T_{net}, as a function of dimension, n, for k-ary n-cubes with 2^8 (256), 2^{14} (16K), and 2^{20} (1M) nodes[5]. The left most data point in this figure corresponds to a torus ($n = 2$) and the right most data point corresponds to a binary n-cube ($k = 2$). This figure assumes constant wire delay, T_c, and a message length, L, of 150 bits. Although constant wire delay is unrealistic, this figure illustrates that even ignoring the dependence of wire delay on wire length, low-dimensional networks achieve lower latency than high-dimensional networks.

The latency of the tori on the left side of Figure 8 is limited almost entirely by distance. The latency of the binary n-cubes on the right side of the graph is limited almost entirely by aspect ratio. With bit serial channels, these cubes take 150 cycles to transmit their messages across a single channel.

In an application that exploits locality of communication, the distance between communicating objects is reduced. In such a situation, the latency of the low dimensional networks (the left side of Figure 8 is reduced. High dimensional networks, on the other hand, cannot take advantage of locality. Their latency will remain high.

In general the lowest latency is achieved when the component of latency due to distance, D, and the component due to message length, $\frac{L}{W}$, are approximately equal, $D \approx \frac{L}{W}$. For the three cases shown in Figure 8, minimum latencies are achieved for $n = 2$, 4, and 5 respectively.

The length of the longest wire in the system, l, becomes a bottleneck that determines the rate at which each channel operates, T_c. The length of this wire is given by

$$l(k,n) = k^{\frac{n}{2}-1}. \tag{13}$$

If the wires are sufficiently short, delay depends logarithmically on wire length. If the channels are longer, they become limited by the speed of light, and delay depends linearly

[5]For the sake of comparison we allow radix to take on non-integer values. For some of the dimensions considered, there is no integer radix, k, that gives the correct number of nodes. In fact, this limitation can be overcome by constructing a *mixed-radix cube*.

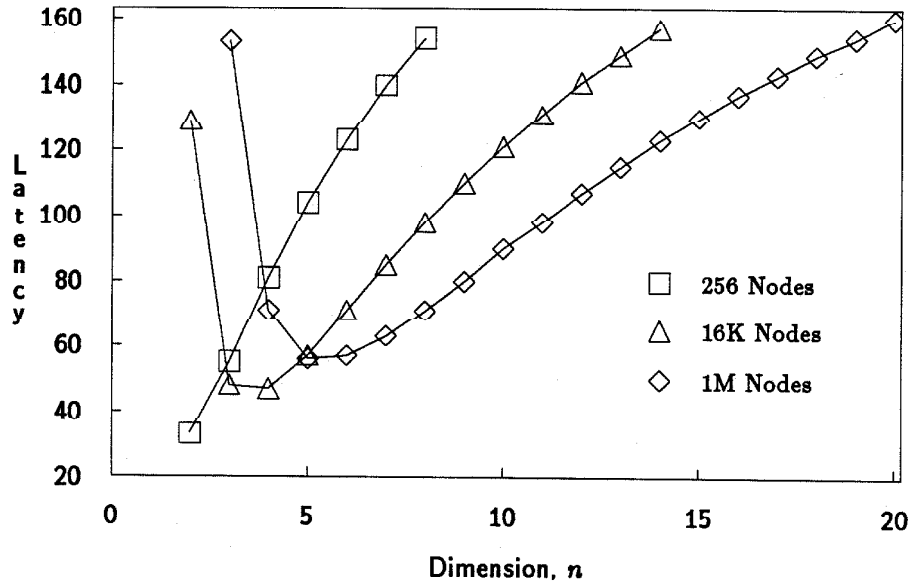

Figure 8: Latency vs. Dimension for 256, 16K, and 1M Nodes, Constant Delay

on channel length. Substituting (13) into (4) and (5) gives

$$T_c \propto \begin{cases} 1 + \log_e l = 1 + \left(\frac{n}{2} - 1\right) \log_e k & \text{logarithmic delay} \\ l = k^{\frac{n}{2}-1} & \text{linear delay.} \end{cases} \tag{14}$$

We substitute (14) into (12) to get the network latency for these two cases:

$$T_l \propto \begin{cases} \left(1 + \left(\frac{n}{2} - 1\right) \log_e k\right) \left(\left(\frac{k-1}{2}\right) n + \frac{2L}{k}\right) & \text{logarithmic delay} \\ \left(k^{\frac{n}{2}-1}\right) \left(\left(\frac{k-1}{2}\right) n + \frac{2L}{k}\right) & \text{linear delay.} \end{cases} \tag{15}$$

Figure 9 shows the average network latency as a function of dimension for k-ary n-cubes with 2^8 (256), 2^{14} (16K), and 2^{20} (1M) nodes, assuming logarithmic wire delay and a message length, L, of 150. Figure 10 shows the same data assuming linear wire delays. In both figures, the left most data point corresponds to a torus ($n = 2$) and the right most data point corresponds to a binary n-cube ($k = 2$).

In the linear delay case, Figure 10, a torus ($n = 2$) always gives the lowest latency. This is because a torus offers the highest bandwidth channels and the most direct physical route between two processing nodes. Under the linear delay assumption, latency is determined solely by bandwidth and by the physical distance traversed. There is no advantage in having long channels.

Under the logarithmic delay assumption, Figure 9, a torus has the lowest latency for small networks ($N = 256$). For the larger networks, the lowest latency is achieved with slightly higher dimensions. With $N = 16K$, the lowest latency occurs when n is three[6]. With $N = 1M$, the lowest latency is achieved when n is 5. It is interesting that assuming constant wire delay does not change this result much. Recall that under the (unrealistic) constant wire delay assumption, Figure 8, the minimum latencies are achieved with dimensions of 2, 4, and 5 respectively.

The results shown in Figures 9 through 8 were derived by comparing networks under the assumption of constant wire cost to a binary n-cube with $W = 1$. For small networks it is possible to construct binary n-cubes with wider channels, and for large networks (e.g., $1M$ nodes) it may not be possible to construct a binary n-cube at all. In the case of small networks, the comparison against binary n-cubes with wide channels can be performed by expressing message length in terms of the binary n-cube's channel width, in effect decreasing the message length for purposes of comparison. The net result is the same: lower-dimensional networks give lower latency. Even if we perform the 256 node comparison against a binary n-cube with $W = 16$, the torus gives the lowest latency under the logarithmic delay model, and a dimension 3 network gives minimum latency under the constant delay model. For large networks, the available wire is less than assumed, so the effective message length should be increased, making low dimensional networks look even more favorable.

[6]In an actual machine the dimension n would be restricted to be an even integer.

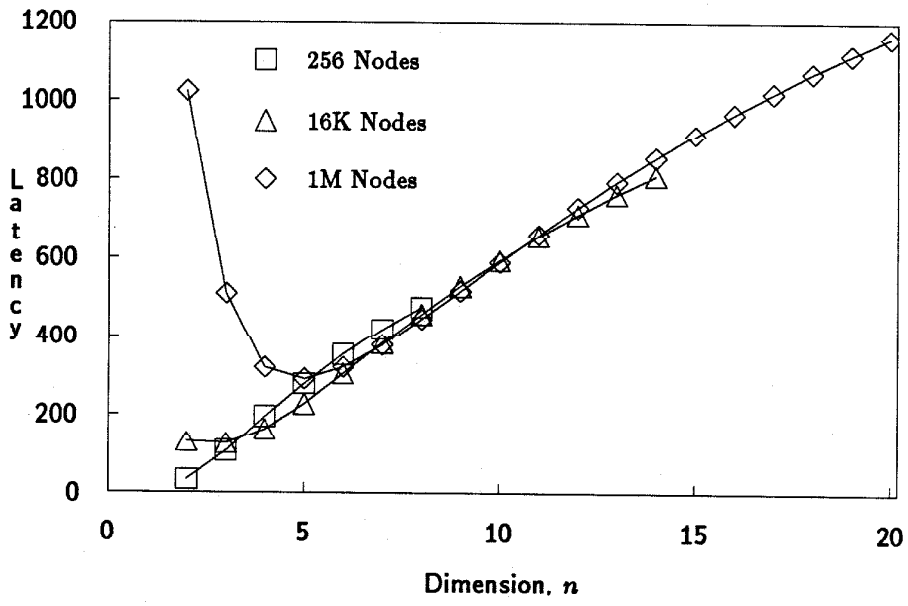

Figure 9: Latency vs. Dimension for 256, 16K, and 1M Nodes, Logarithmic Delay

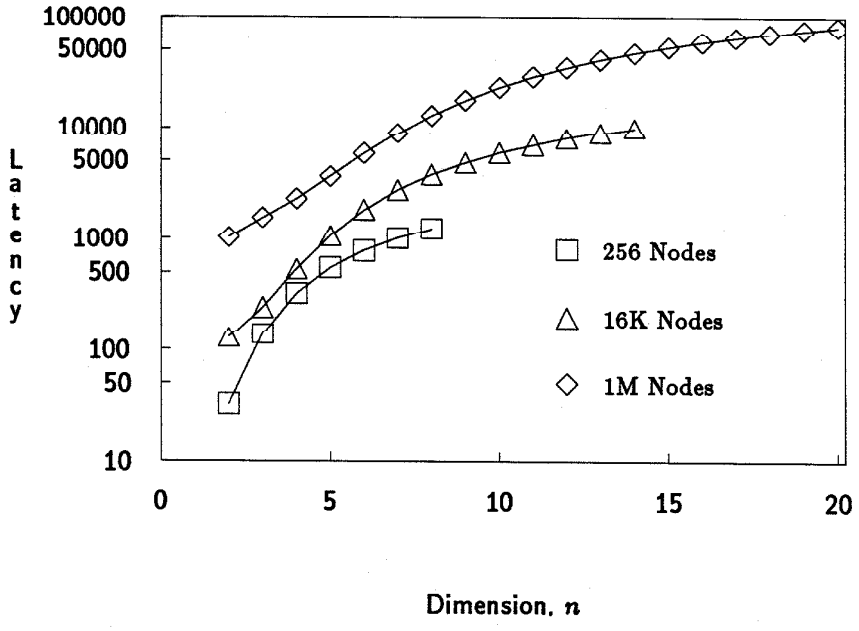

Figure 10: Latency vs. Dimension for 256, 16K, and 1M Nodes, Linear Delay

In this comparison we have assumed that only a single bit of information is in transit on each wire of the network at a given time. Under this assumption, the delay between nodes, T_c, is equal to the period of each node, T_p. In a network with long wires, however, it is possible to have several bits in transit at once. In this case, the channel delay, T_c, is a function of wire length, while the channel period, $T_p < T_c$, remains constant. Similarly, in a network with very short wires we may allow a bit to ripple through several channels before sending the next bit. In this case, $T_p > T_c$. Separating the coefficients, T_c and T_p, (3) becomes

$$T_{\text{net}} = \left(T_c D + T_p \frac{L}{W} \right). \tag{16}$$

The net effect of allowing $T_c \neq T_p$ is the same as changing the length, L, by a factor of $\frac{T_p}{T_c}$ and does not change our results significantly.

When wire cost is considered, low-dimensional networks (e.g., tori) offer lower latency than high-dimensional networks (e.g., binary n-cubes). Intuitively, tori outperform binary n-cubes because they better match form to function. The logical and physical graphs of the torus are identical; Thus, messages always travel the minimum distance from source to destination. In a binary n-cube, on the other hand, the fit between form and function is not as good. A message in a binary n-cube embedded into the plane may have to traverse considerably more than the minimum distance between its source and destination.

5.2 Throughput

Throughput, another important metric of network performance, is defined as the total number of messages the network can handle per unit time. One method of estimating throughput is to calculate the capacity of a network, the total number of messages that can be in the network at once. Typically the maximum throughput of a network is some fraction of its capacity. The network capacity per node is the total bandwidth out of each node divided by the average number of channels traversed by each message. For k-ary n-cubes, the bandwidth out of each node is nW, and the average number of channels traversed is given by (11), so the network capacity per node is given by

$$\Gamma(k,n) \propto \frac{nW(k,n)}{D(k,n)} \propto \frac{n\left(\frac{k}{2}\right)}{\left(\frac{k-1}{2}\right)n} \approx 1. \tag{17}$$

The network capacity is independent of dimension. For a constant wire density, there is a constant network capacity.

To verify that throughput is proportional to capacity and to determine the constant of proportionality, I measured the throughput of k-ary n-cubes with 256 and 4K nodes using a k-ary n-cube network simulator. The simulations were performed using random traffic and the deterministic routing algorithm described in [6]. The traffic was generated by repeatedly having each node randomly choose a destination node and initiate transmission of a message to the destination node.

The results of the throughput experiment are tabulated below. Throughput is shown as a fraction of capacity. The experimental results indicate that throughput is between 0.1

Parameter	256 Nodes			4K Nodes			
Dimension	2	4	8	2	4	6	12
radix	16	4	2	64	8	4	2
Throughput	0.468	0.618	0.596	0.492	0.426	0.427	0.460

Table 1: Throughput as a fraction of network capacity.

and 0.7 capacity for the cases tested. For practical purposes throughput is independent of dimension.

5.3 Hot Spot Throughput

In many situations traffic is not uniform, but rather is concentrated into *hot spots*. A *hot spot* is a pair of nodes that accounts for a disproportionately large portion of the total network traffic. As described by Pfister [13] for a shared-memory computer, hot-spot traffic can degrade performance of the entire network by causing congestion.

The *hot-spot throughput* of a network is the maximum rate at which messages can be sent from one specific node, P_i, to another specific node, P_j. For a k-ary n-cube with deterministic routing, the hot-spot throughput, Θ_{HS}, is just the bandwidth of a single channel, W. Thus, under the assumption of constant wire cost we have

$$\Theta_{HS} = W = k - 1. \tag{18}$$

Low-dimensional networks have greater channel bandwidth and thus have greater hot-spot throughput than do high-dimensional networks. Intuitively, low-dimensional networks operate better under non-uniform loads because they do more resource sharing. In an interconnection network the resources are wires. In a high-dimensional network, wires are assigned to particular dimensions and cannot be shared between dimensions. For example, in a binary n-cube it is possible for a wire to be saturated while a physically adjacent wire assigned to a different dimension remains idle. In a torus all physically adjacent wires are combined into a single channel that is shared by all messages that must traverse the physical distance spanned by the channel.

Bibliography

[1] Batcher, K.E., "Sorting Networks and Their Applications," *Proceedings AFIPS FJCC,* Vol. 32, 1968, pp. 307-314.

[2] Batcher, K.E., "The Flip Network in STARAN," *Proceedings, 1976 International Conference on Parallel Processing,* pp. 65-71.

[3] Benes, V.E., *Mathematical Theory of Connecting Networks and Telephone Traffic,* Academic, New York, 1965.

[4] Browning, Sally, *The Tree Machine: A Highly Concurrent Computing Environment,* Dept. of Computer Science, California Institute of Technology, Technical Report 3760, 1985.

[5] Dally, William J., *A VLSI Architecture for Concurrent Data Structures,* Ph.D. Thesis, Department of Computer Science, California Institute of Technology, Technical Report 5209:TR:86, 1986.

[6] Dally, William J. and Seitz, Charles L., *Deadlock-Free Message Routing in Multiprocessor Interconnection Networks,* Dept. of Computer Science, California Institute of Technology, Technical Report 5206:TR:86, 1986.

[7] Dally, William J. and Seitz, Charles L., "The Torus Routing Chip," to appear in *J. Distributed Systems,* Vol. 1, No. 3, 1986.

[8] Kermani, Parviz and Kleinrock, Leonard, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks,* Vol 3., 1979, pp. 267-286.

[9] Lawrie, Duncan H., "Alignment and Access of Data in an Array Processor," *IEEE Transactions on Computers,* Vol. C-24, No. 12, December 1975, pp. 1145-1155.

[10] Leiserson, Charles, L., "Fat Trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Transactions on Computers,* Vol. C-34, No. 10, October 1985, pp. 892-901.

[11] Mead, Carver A. and Conway, Lynn A., *Introduction to VLSI Systems,* Addison-Wesley, Reading, Mass., 1980.

[12] Pease, M.C., III, "The Indirect Binary n-Cube Microprocessor Array," *IEEE Transactions on Computers,* Vol. C-26, No. 5, May 1977, pp. 458-473.

[13] Pfister, G.F. and Norton, V.A., "Hot Spot Contention and Combining in Multistage Interconnection Networks," *IEEE Transactions on Computers,* Vol. C-34, No. 10, October 1985, pp. 943-948.

[14] Seigel, Howard Jay, "Interconnection Networks for SIMD Machines," *IEEE Computer,* Vol. 12, No. 6, June 1979, pp. 57-65.

[15] Seitz, Charles L., "Concurrent VLSI Architectures," *IEEE Transactions on Computers,* Vol. C-33, No. 12, December 1984, pp. 1247-1265.

[16] Seitz, Charles L., et al., *The Hypercube Communications Chip,* Dept. of Computer Science, California Institute of Technology, Display File 5182:DF:85, March 1985.

[17] Sequin, Carlo, H., "Single Chip Computers, The New VLSI Building Block," *Caltech Conference on VLSI,* C. L. Seitz Ed. January 1979, pp. 435-452.

[18] Stone, H.S., "Parallel Processing with the Perfect Shuffle," *IEEE Transactions on Computers,* Vol. C-20, No. 2, February 1971, pp. 153-161.

[19] Sullivan, H. and Bashkow, T.R., "A Large Scale Homogeneous Machine," *Proc. 4th Annual Symposium on Computer Architecture,* 1977, pp. 105-124.

[20] Tanenbaum, A. S., *Computer Networks,* Prentice Hall, Englewood Cliffs, N.J., 1981.

[21] Thompson, C.D., *A Complexity Theory of VLSI,* Department of Computer Science, Carnegie-Mellon University, Technical Report CMU-CS-80-140, August 1980.